PORTAL
USPTO

Subscribe (Full Service) Register (Limited Service, Free) Login

Search: [✓ The ACM Digital Library] The Guide

+rewrite +bytecode +static +initializer

THE ACM DIGITAL LIBRARY

Feedback Report a problem Satisfaction survey

Terms used **rewrite** **bytecode** **static** **initializer**

Found **14** of **157,873**

Sort results by: relevance

Display results: expanded form

Save results to a Binder

Search Tips

Open results in a new window

Try an Advanced Search
Try this search in The ACM Guide

Results 1 - 14 of 14

Relevance scale

1 Portable resource control in Java
Walter Binder, Jane G. Hulaas, Alex Villazón
October 2001 **ACM SIGPLAN Notices , Proceedings of the 16th ACM SIGPLAN conference on Object oriented programming, systems, languages, and applications**, Volume 36 Issue 11

Full text available: pdf(307.08 KB) Additional Information: full citation, abstract, references, citings, index terms

Preventing abusive resource consumption is indispensable for all kinds of systems that execute untrusted mobile coee, such as mobile object sytems, extensible web servers, and web browsers. To implement the required defense mechanisms, some support for resource control must be available: accounting and limiting the usage of physical resources like CPU and memory, and of logical resources like threads. Java is the predominant implementation language for the kind of systems envisaged here, even th ...

Keywords: Java, bytecode rewriting, micro-kernels, mobile object systems, resource control, security

2 Application isolation in the Java Virtual Machine
Grzegorz Czajkowski
October 2000 **ACM SIGPLAN Notices , Proceedings of the 15th ACM SIGPLAN conference on Object-oriented programming, systems, languages, and applications**, Volume 35 Issue 10

Full text available: pdf(217.49 KB) Additional Information: full citation, abstract, references, citings, index terms

To date, systems offering multitasking for the Java&trade; programming language either use one process or one class loader for each application. Both approaches are unsatisfactory. Using operating system processes is expensive, scales poorly and does not fully exploit the protection features inherent in a safe language. Class loaders replicate application code, obscure the type system, and non-uniformly treat 'trusted' and 'untrusted' classes, which leads to subtle, but nevertheless, potenti ...

Keywords: Java Virtual Machine, application isolation, multitasking

3 Multitasking without comprimise: a virtual machine evolution
Grzegorz Czajkowski, Laurent Daynés

October 2001 **ACM SIGPLAN Notices , Proceedings of the 16th ACM SIGPLAN conference on Object oriented programming, systems, languages, and applications**, Volume 36 Issue 11

Full text available: pdf(220.97 KB) Additional Information: full citation, abstract, references, citings, index terms

The multitasking virtual machine (called from now on simply MVM) is a modification of the Java virtual machine. It enables safe, secure, and scalable multitasking. Safety is achieved by strict isolation of application from one another. Resource control augment security by preventing some denial-of-service attacks. Improved scalability results from an aggressive application of the main design principle of MVM: share as much of the runtime as possible among applications and replicate everything el ...

Keywords: Java virtual machine, application isolation, native code execution, resource control

4 SAFKASI: a security mechanism for language-based systems
Dan S. Wallach, Andrew W. Appel, Edward W. Felten
October 2000 **ACM Transactions on Software Engineering and Methodology (TOSEM)**, Volume 9 Issue 4

Full text available: pdf(234.89 KB) Additional Information: full citation, abstract, references, citings, index terms

In order to run untrusted code in the same process as trusted code, there must be a mechanism to allow dangerous calls to determine if their caller is authorized to exercise the privilege of using the dangerous routine. Java systems have adopted a technique called stack inspection to address this concern. But its original definition, in terms of searching stack frames, had an unclear relationship to the actual achievement of security, overconstrained the implementation of a Java system, lim ...

Keywords: Internet, Java, WWW, access control, applets, security-passing style, stack inspection

5 A practical type system and language for reference immutability
Adrian Birka, Michael D. Ernst
October 2004 **ACM SIGPLAN Notices , Proceedings of the 19th annual ACM SIGPLAN Conference on Object-oriented programming, systems, languages, and applications**, Volume 39 Issue 10

Full text available: pdf(171.73 KB) Additional Information: full citation, abstract, references, index terms

This paper describes a type system that is capable of expressing and enforcing immutability constraints. The specific constraint expressed is that the abstract state of the object to which an immutable reference refers cannot be modified using that reference. The abstract state is (part of) the transitively reachable state: that is, the state of the object and all state reachable from it by following references. The type system permits explicitly excluding fields or objects from the abstract ...

Keywords: Java, Javari, const, immutability, mutable, readonly, type system, verification

6 Agents, interactions, mobility and systems: Secure mobile agent systems using Java: where are we heading?
Walter Binder, Volker Roth
March 2002 **Proceedings of the 2002 ACM symposium on Applied computing**

Full text available: pdf(501.91 KB) Additional Information: full citation, abstract, references, index terms

Java is the predominant language for mobile agent systems, both for implementing mobile agent execution environments and for writing mobile agent applications. This is due to inherent support for code mobility by means of dynamic class loading and separable class name spaces, as well as a number of security properties, such as language safety and access control by means of stack introspection. However, serious questions must be raised whether Java is actually up to the task of providing a secure ...

Keywords: Java, mobile agents, security, survey

7 A comparative study of static and profile-based heuristics for inlining
Matthew Arnold, Stephen Fink, Vivek Sarkar, Peter F. Sweeney
January 2000 **ACM SIGPLAN Notices , Proceedings of the ACM SIGPLAN workshop on Dynamic and adaptive compilation and optimization**, Volume 35 Issue 7
Full text available: pdf(1.13 MB) Additional Information: full citation, abstract, references, citings, index terms

In this paper, we present a comparative study of static and profile-based heuristics for inlining. Our motivation for this study is to use the results to design the best inlining algorithm that we can for the Jalapeño dynamic optimizing compiler for Java [6]. We use a well-known approximation algorithm for the KNAPSACK problem as a common "meta-algorithm" for the inlining heuristics studied in this paper. We present performance results for an implementation of these inlinin ...

8 Practical extraction techniques for Java
Frank Tip, Peter F. Sweeney, Chris Laffra, Aldo Eisma, David Streeter
November 2002 **ACM Transactions on Programming Languages and Systems (TOPLAS)**, Volume 24 Issue 6
Full text available: pdf(1.01 MB) Additional Information: full citation, abstract, references, citings, index terms, review

Reducing application size is important for software that is distributed via the internet, in order to keep download times manageable, and in the domain of embedded systems, where applications are often stored in (Read-Only or Flash) memory. This paper explores extraction techniques such as the removal of unreachable methods and redundant fields, inlining of method calls, and transformation of the class hierarchy for reducing application size. We implemented a number of extraction techniques in < ...

Keywords: Application extraction, call graph construction, class hierarchy transformation, packaging, whole-program analysis

9 Program transformations for portable CPU accounting and control in Java
Jarle Hulaas, Walter Binder
August 2004 **Proceedings of the 2004 ACM SIGPLAN symposium on Partial evaluation and semantics-based program manipulation**
Full text available: pdf(220.22 KB) Additional Information: full citation, abstract, references, index terms

In this paper we introduce a novel scheme for portable CPU accounting and control in Java, which is based on program transformation techniques at the bytecode level and can be used with every standard Java Virtual Machine. In our approach applications, middleware, and the standard java runtime libraries (i.e., the Java Development Kit, or JDK) are modified in order to expose details regarding the execution of threads. This paper presents the details of how we re-engineer Java bytecode for CPU ma ...

Keywords: Java, bytecode engineering, program transformations, resource management

10 Document querying and transformation: Lazy XSL transformations
Steffen Schott, Markus L. Noga
November 2003 **Proceedings of the 2003 ACM symposium on Document engineering**
Full text available: pdf(335.83 KB) Additional Information: full citation, abstract, references, index terms

We introduce a lazy XSLT interpreter that provides random access to the transformation result. This allows efficient pipelining of transformation sequences. Nodes of the result tree are computed only upon initial access. As these computations have limited fan-in, sparse output coverage propagates backwards through the pipeline.In comparative measurements with traditional eager implementations, our approach is on par for complete coverage and excels as coverage becomes sparser. In contrast to eag ...

11 Maya: multiple-dispatch syntax extension in Java
Jason Baker, Wilson C. Hsieh
May 2002 **ACM SIGPLAN Notices , Proceedings of the ACM SIGPLAN 2002 Conference on Programming language design and implementation**, Volume 37 Issue 5
Full text available: pdf(152.75 KB) Additional Information: full citation, abstract, references, citings, index terms

We have designed and implemented Maya, a version of Java that allows programmers to extend and reinterpret its syntax. Maya generalizes macro systems by treating grammar productions as generic functions, and semantic actions on productions as multimethods on the corresponding generic functions. Programmers can write new generic functions (i.e., grammar productions) and new multimethods (i.e., semantic actions), through which they can extend the grammar of the language and change the semantics of ...

Keywords: Java, generative programming, macros, metaprogramming

12 Extending Java for high-level Web service construction
Aske Simon Christensen, Anders Møller, Michael I. Schwartzbach
November 2003 **ACM Transactions on Programming Languages and Systems (TOPLAS)**, Volume 25 Issue 6
Full text available: pdf(947.02 KB) Additional Information: full citation, abstract, references, citings, index terms

We incorporate innovations from the <bigwig> project into the Java language to provide high-level features for Web service programming. The resulting language, JWIG, contains an advanced session model and a flexible mechanism for dynamic construction of XML documents, in particular XHTML. To support program development we provide a suite of program analyses that at compile time verify for a given program that no runtime errors can occur while building documents or receiving form input, and ...

Keywords: Interactive Web services, XML, data-flow analysis

13 Technical papers: dynamic program analysis: Role-based exploration of object-oriented programs
Brian Demsky, Martin Rinard
May 2002 **Proceedings of the 24th International Conference on Software Engineering**
Full text available: pdf(1.24 MB) Additional Information: full citation, abstract, references, citings, index terms

We present a new technique for helping developers understand heap properties of object-oriented programs and how the actions of the program affect these properties. Our dynamic analysis uses the aliasing properties of objects to synthesize a set of *roles;* each role represents an abstract object state intended to be of interest to the developer. We allow the developer to customize the analysis to explore the object states and behavior of the program at multiple different and potentially co ...

14 Extracting library-based object-oriented applications
Peter F. Sweeney, Frank Tip
November 2000 **ACM SIGSOFT Software Engineering Notes , Proceedings of the 8th ACM SIGSOFT international symposium on Foundations of software engineering: twenty-first century applications**, Volume 25 Issue 6

Full text available: pdf(1.06 MB) Additional Information: full citation, abstract, references, citings, index terms

In an increasingly popular model of software distribution, software is developed in one computing environment and deployed in other environments by transfer over the internet. Extraction tools perform a static whole-program analysis to determine unused functionality in applications in order to reduce the time required to download applications. We have identified a number of scenarios where extraction tools require information beyond what can be inferred through static analysis: software distr ...

Results 1 - 14 of 14

PORTAL
USPTO

Subscribe (Full Service) Register (Limited Service, Free) Login

Search: [✓ The ACM Digital Library] The Guide

+rewrite +bytecode +scalar

THE ACM DIGITAL LIBRARY

Feedback Report a problem Satisfaction survey

Terms used **rewrite bytecode scalar** Found **23** of **157,873**

Sort results by: relevance
Display results: expanded form

Save results to a Binder
Search Tips
Open results in a new window

Try an Advanced Search
Try this search in The ACM Guide

Results 1 - 20 of 23 Result page: **1** 2 next

Relevance scale

1 Engineering a customizable intermediate representation
K. Palacz, J. Baker, C. Flack, C. Grothoff, H. Yamauchi, J. Vitek
June 2003 **Proceedings of the 2003 workshop on Interpreters, virtual machines and emulators**
Full text available: pdf(322.87 KB) Additional Information: full citation, abstract, references, citings

The Ovm framework is a set of tools and components for building language runtimes. We present the intermediate representation and software design patterns used throughout the framework. One of the main themes in this work has been to support experimentation with new linguistic constructs and implementation techniques. To this end, framework components were designed to be parametric with respect to the instruction set on which they operate. We argue that our approach eases the task of writing new ...

2 Automatic translation of Fortran to JVM bytecode
Keith Seymour, Jack Dongarra
June 2001 **Proceedings of the 2001 joint ACM-ISCOPE conference on Java Grande**
Full text available: pdf(555.04 KB) Additional Information: full citation, abstract, references, index terms

This paper reports on the design of a FORTRAN-to-Java translator whose target language is the instruction set of the Java Virtual Machine. The goal of the translator is to generate Java implementations of legacy FORTRAN numerical codes in a consistent and reliable fashion. The benefits of directly generating bytecode are twofold. First, it provides a much more straightforward and efficient mechanism for translating FORTRAN GOTO statements. Second, it provides a framework for pursuing various ...

3 Instrumentation of standard libraries in object-oriented languages: the twin class hierarchy approach
Michael Factor, Assaf Schuster, Konstantin Shagin
October 2004 **ACM SIGPLAN Notices , Proceedings of the 19th annual ACM SIGPLAN Conference on Object-oriented programming, systems, languages, and applications**, Volume 39 Issue 10
Full text available: pdf(227.01 KB) Additional Information: full citation, abstract, references, index terms

Code instrumentation is widely used for a range of purposes that include profiling, debugging, visualization, logging, and distributed computing. Due to their special status within the language infrastructure, the <i>standard class libraries</i>, also known as <i>system classes</i> provided by most contemporary object-oriented languages are difficult and sometimes impossible to instrument. If instrumented, the use of their rewritten versions within the instrumentation code is ...

Keywords: code instrumentation, inheritance, java, standard class libraries

4 Automatic pool allocation for disjoint data structures
Chris Lattner, Vikram Adve
June 2002 **ACM SIGPLAN Notices , Proceedings of the 2002 workshop on Memory system performance**, Volume 38 Issue 2 supplement
Full text available: pdf(1.48 MB) Additional Information: full citation, abstract, references, citings

This paper presents an analysis technique and a novel program transformation that can enable powerful optimizations for entire linked data structures. The fully automatic transformation converts ordinary programs to use pool (aka region) allocation for heap-based data structures. The transformation relies on an efficient link-time interprocedural analysis to identify disjoint data structures in the program, to check whether these data structures are accessed in a type-safe manner, and to constru ...

5 A study of devirtualization techniques for a Java Just-In-Time compiler
Kazuaki Ishizaki, Motohiro Kawahito, Toshiaki Yasue, Hideaki Komatsu, Toshio Nakatani
October 2000 **ACM SIGPLAN Notices , Proceedings of the 15th ACM SIGPLAN conference on Object-oriented programming, systems, languages, and applications**, Volume 35 Issue 10
Full text available: pdf(225.89 KB) Additional Information: full citation, abstract, references, citings, index terms

Many devirtualization techniques have been proposed to reduce the runtime overhead of dynamic method calls for various object-oriented languages, however, most of them are less effective or cannot be applied for Java in a straightforward manner. This is partly because Java is a statically-typed language and thus transforming a dynamic call to a static one does not make a tangible performance gain (owing to the low overhead of accessing the method table) unless it is inlined, and partly because t ...

6 Implementing jalapeño in Java
Bowen Alpern, C. R. Attanasio, Anthony Cocchi, Derek Lieber, Stephen Smith, Ton Ngo, John J. Barton, Susan Flynn Hummel, Janice C. Sheperd, Mark Mergen
October 1999 **ACM SIGPLAN Notices , Proceedings of the 14th ACM SIGPLAN conference on Object-oriented programming, systems, languages, and applications**, Volume 34 Issue 10
Full text available: pdf(1.57 MB) Additional Information: full citation, abstract, references, citings, index terms

Jalapeño is a virtual machine for Java™ servers written in Java.A running Java program involves four layers of functionality: the user code, the virtual-machine, the operating system, and the hardware. By drawing the Java / non-Java boundary below the virtual machine rather than above it, Jalapeño reduces the boundary-crossing overhead and opens up more opportunities for optimization.To get Jalapeño started, a boot image of a ...

7 Dynamic Adaptive compilation: Design, implementation and evaluation of adaptive recompilation with on-stack replacement
Stephen J. Fink, Feng Qian
March 2003 **Proceedings of the international symposium on Code generation and optimization: feedback-directed and runtime optimization CGO '03**
Full text available: pdf(1.02 MB) Additional Information: full citation, abstract, references, citings, index terms

Modern virtual machines often maintain multiple compiled versions of a method. An on-stack replacement (OSR) mechanism enables a virtual machine to transfer execution between compiled versions, even while a method runs. Relying on this mechanism, the system can exploit powerful techniques to reduce compile time and code space, dynamically de-optimize code, and invalidate speculative optimizations.This paper presents a new, simple, mostly compiler-independent mechanism to transfer execution into ...

8 Adding tuples to Java: a study in lightweight data structures
C. van Reeuwijk, H. J. Sips
November 2002 **Proceedings of the 2002 joint ACM-ISCOPE conference on Java Grande**

Full text available: pdf(91.24 KB) Additional Information: full citation, abstract, references, citings, index terms

Java classes are very flexible, but this comes at a price. The main cost is that every class instance must be dynamically allocated. Their access by reference introduces pointer dereferences and complicates program analysis. These costs are particularly burdensome for small, ubiquitous data structures such as coordinates and state vectors. For such data structures a *lightweight* representation is desirable, allowing such data to be handled directly, similar to primitive types. A number of ...

Keywords: Java, lightweight data structures, tuple

9 Partial method compilation using dynamic profile information
John Whaley
October 2001 **ACM SIGPLAN Notices , Proceedings of the 16th ACM SIGPLAN conference on Object oriented programming, systems, languages, and applications**, Volume 36 Issue 11

Full text available: pdf(1.73 MB) Additional Information: full citation, abstract, references, citings, index terms

The traditional tradeoff when performing dynamic compilation is that of fast compilation time versus fast code performance. Most dynamic compilation systems for Java perform selective compilation and/or optimization at a method granularity. This is the not the optimal granularity level. However, compiling at a sub-method granularity is thought to be too complicated to be practical. This paper describes a straightforward technique for performing compilation and optimizations at a finer, sub-metho ...

10 Research papers I: Interpreting programs in static single assignment form
Jeffery von Ronne, Ning Wang, Michael Franz
June 2004 **Proceedings of the 2004 workshop on Interpreters, virtual machines and emulators**

Full text available: pdf(165.13 KB) Additional Information: full citation, abstract, references

Optimizing compilers, including those in virtual machines, commonly utilize Static Single Assignment Form as their intermediate representation, but interpreters typically implement stack-oriented virtual machines. This paper introduces an easily interpreted variant of Static Single Assignment Form. Each instruction of this Interpretable Static Single Assignment Form, including the Phi Instruction, has self-contained operational semantics facilitating efficient interpretation. Even the array mani ...

11 Embedded systems: applications, solutions and techniques (EMBS): Automatic generation of application-specific systems based on a micro-programmed Java core
F. Gruian, P. Andersson, K. Kuchcinski, M. Schoeberl
March 2005 **Proceedings of the 2005 ACM symposium on Applied computing**

Full text available: pdf(331.35 KB) Additional Information: full citation, abstract, references, index terms

This paper describes a co-design based approach for automatic generation of application specific systems, suitable for FPGA-centric embedded applications. The approach augments a processor core with hardware accelerators extracted automatically from a high-level specification (Java) of the application, to obtain a custom system, optimised for the target application. We advocate herein the use of a microprogrammed core as the basis for system generation in order to hide the hardware access operat ...

Keywords: FPGA, Java, co-design, system-on-chip

12 A dynamic optimization framework for a Java just-in-time compiler
Toshio Suganuma, Toshiaki Yasue, Motohiro Kawahito, Hideaki Komatsu, Toshio Nakatani
October 2001 **ACM SIGPLAN Notices , Proceedings of the 16th ACM SIGPLAN conference on Object oriented programming, systems, languages, and applications**, Volume 36 Issue 11
Full text available: pdf(2.12 MB) Additional Information: full citation, abstract, references, citings, index terms

The high performance implementation of Java Virtual Machines (JVM) and just-in-time (JIT) compilers is directed toward adaptive compilation optimizations on the basis of online runtime profile information. This paper describes the design and implementation of a dynamic optimization framework in a production-level Java JIT compiler. Our approach is to employ a mixed mode interpreter and a three level optimizing compiler, supporting quick, full, and special optimization, each of which has a differ ...

13 Design, implementation, and evaluation of optimizations in a just-in-time compiler
Kazuaki Ishizaki, Motohiro Kawahito, Toshiaki Yasue, Mikio Takeuchi, Takeshi Ogasawara, Toshio Suganuma, Tamiya Onodera, Hideaki Komatsu, Toshio Nakatani
June 1999 **Proceedings of the ACM 1999 conference on Java Grande**
Full text available: pdf(1.09 MB) Additional Information: full citation, references, citings, index terms

14 Enhancing software reliability with speculative threads
Jeffrey Oplinger, Monica S. Lam
October 2002 **Proceedings of the 10th international conference on Architectural support for programming languages and operating systems**, Volume 37 , 36 , 30 Issue 10 , 5 , 5
Full text available: pdf(1.47 MB) Additional Information: full citation, abstract, references, citings

This paper advocates the use of a monitor-and-recover programming paradigm to enhance the reliability of software, and proposes an architectural design that allows software and hardware to cooperate in making this paradigm more efficient and easier to program.We propose that programmers write monitoring functions assuming simple sequential execution semantics. Our architecture speeds up the computation by executing the monitoring functions speculatively in parallel with the main computation. For ...

15 Dynamic compilation techniques: Optimized interval splitting in a linear scan register allocator
Christian Wimmer, Hanspeter Mössenböck
June 2005 **Proceedings of the 1st ACM/USENIX international conference on Virtual execution environments**
Full text available: pdf(341.54 KB) Additional Information: full citation, abstract, references, index terms

We present an optimized implementation of the linear scan register allocation algorithm for Sun Microsystems' Java HotSpot™ client compiler. Linear scan register allocation is especially suitable for just-in-time compilers because it is faster than the common graph-coloring approach and yields results of nearly the same quality.Our allocator improves the basic linear scan algorithm by adding more advanced optimizations: It makes use of lifetime holes, splits intervals if the register press ...

Keywords: compilers, graph-coloring, java, just-in-time compilation, linear scan, optimization, register allocation

16 A comparative study of static and profile-based heuristics for inlining
Matthew Arnold, Stephen Fink, Vivek Sarkar, Peter F. Sweeney
January 2000 **ACM SIGPLAN Notices , Proceedings of the ACM SIGPLAN workshop on Dynamic and adaptive compilation and optimization**, Volume 35 Issue 7
Full text available: pdf(1.13 MB) Additional Information: full citation, abstract, references, citings, index terms

In this paper, we present a comparative study of static and profile-based heuristics for inlining. Our motivation for this study is to use the results to design the best inlining algorithm that we can for the Jalapeño dynamic optimizing compiler for Java [6]. We use a well-known approximation algorithm for the KNAPSACK problem as a common "meta-algorithm" for the inlining heuristics studied in this paper. We present performance results for an implementation of these inlinin ...

17 Efficient incremental run-time specialization for free
Renaud Marlet, Charles Consel, Philippe Boinot
May 1999 **ACM SIGPLAN Notices , Proceedings of the ACM SIGPLAN 1999 conference on Programming language design and implementation**, Volume 34 Issue 5
Full text available: pdf(1.36 MB) Additional Information: full citation, abstract, references, citings, index terms

Availability of data in a program determines computation stages. Incremental partial evaluation exploit these stages for optimization: it allows further specialization to be performed as data become available at later stages. The fundamental advantage of incremental specialization is to factorize the specialization process. As a result, specializing a program at a given stage costs considerably less than specializing it once all the data are available.We present a realistic and flexible approach ...

18 The structure and performance of interpreters
Theodore H. Romer, Dennis Lee, Geoffrey M. Voelker, Alec Wolman, Wayne A. Wong, Jean-Loup Baer, Brian N. Bershad, Henry M. Levy
September 1996 **Proceedings of the seventh international conference on Architectural support for programming languages and operating systems**, Volume 31 , 30 Issue 9 , 5
Full text available: pdf(1.17 MB) Additional Information: full citation, abstract, references, citings, index terms

Interpreted languages have become increasingly popular due to demands for rapid program development, ease of use, portability, and safety. Beyond the general impression that they are "slow," however, little has been documented about the performance of interpreters as a class of applications.This paper examines interpreter performance by measuring and analyzing interpreters from both software and hardware perspectives. As examples, we measure the MIPSI, Java, Perl, and Tcl interpreters running an ...

19 Specialization tools and techniques for systematic optimization of system software
Dylan McNamee, Jonathan Walpole, Calton Pu, Crispin Cowan, Charles Krasic, Ashvin Goel, Perry Wagle, Charles Consel, Gilles Muller, Renauld Marlet
May 2001 **ACM Transactions on Computer Systems (TOCS)**, Volume 19 Issue 2
Full text available: pdf(178.52 KB) Additional Information: full citation, abstract, references, citings, index terms

Specialization has been recognized as a powerful technique for optimizing operating systems. However, specialization has not been broadly applied beyond the research community because current techniques based on manual specialization, are time-consuming and error-prone. The goal of the work described in this paper is to help operating system tuners perform specialization more easily. We have built a specialization toolkit that assists the major tasks of specializing operating systems. We de ...

Keywords: operating system specialization, optimization, software architecture

20 Real-time techniques: Static determination of allocation rates to support real-time garbage collection
Tobias Mann, Morgan Deters, Rob LeGrand, Ron K. Cytron
June 2005 **Proceedings of the 2005 ACM SIGPLAN/SIGBED conference on Languages, compilers, and tools for embedded systems**
Full text available: pdf(175.91 KB) Additional Information: full citation, abstract, references, index terms

While it is generally accepted that garbage-collected languages offer advantages over languages in which objects must be explicitly deallocated, real-time developers are leery of the adverse effects a garbage collector might have on real-time performance. Semiautomatic approaches based on regions have been proposed, but incorrect usage could cause unbounded storage leaks or program failure. Moreover, correct usage cannot be guaranteed at compile time. Recently, real-time garbage collectors have ...

Keywords: allocation rate, real-time garbage collection, static analysis

Results 1 - 20 of 23 Result page: **1** 2 next